Middleware and Web Services for the Collaborative Information Portal of NASA's Mars Exploration Rovers Mission

Elias Sinderson, Vish Magapu, Ronald Mak

NASA Ames Research Center, M/S 269-3
Moffett Field, California 94035
{esinderson, vmagapu, rmak}@mail.arc.nasa.gov

Abstract. We describe the design and deployment of the middleware for the Collaborative Information Portal (CIP), a mission critical J2EE application developed for NASA's 2003 Mars Exploration Rover mission. CIP enabled mission personnel to access data and images sent back from Mars, staff and event schedules, broadcast messages and clocks displaying various Earth and Mars time zones. We developed the CIP middleware in less than two years time using cutting-edge technologies, including EJBs, servlets, JDBC, JNDI and JMS. The middleware was designed as a collection of independent, hot-deployable web services, providing secure access to back end file systems and databases. Throughout the middleware we enabled crosscutting capabilities such as runtime service configuration, security, logging and remote monitoring. This paper presents our approach to mitigating the challenges we faced, concluding with a review of the lessons we learned from this project and noting what we'd do differently and why.

1 Introduction

The 2003 Mars Exploration Rover (MER) mission was the latest in a series of science missions to the planet Mars. The primary goal was to search for irrefutable evidence of liquid water existing on the surface in the Martian past. The mission was composed of two redundant (dual launch, dual lander) mobile science platforms, or *rovers*, outfitted with a variety of instruments. NASA launched two spacecraft carrying these rovers in June 2003, and in January 2004 they landed safely on Mars. After landing the rovers at two separate locations, they proceeded to take numerous images of their surroundings and collect data from the Martian rocks and soil. Intermittently throughout each Martian day, or sol, the rovers transmitted their data back to Earth where it was collaboratively analyzed and activities were planned for the next sol. This analysis and planning cycle had to be completed in time for the next set of command sequences to be uplinked as the sun is rising on Mars.

The MER Collaborative Information Portal (CIP) was conceived of to provide mission personnel a sense of situational awareness, allowing the MER scientists,

engineers and managers to accomplish their daily tasks by facilitating access to various mission data products, schedules, broadcast announcements, and the like. This paper focuses on the middleware and web services that were developed for CIP. The rest of this section presents a brief overview of the day-to-day mission operations so as to lay a conceptual foundation for the work described in the rest of the paper. Section 2 outlines the overall approach to systems development that was employed, including the development environment and utilities we selected. Section 3 provides an initial, high-level overview of the client application functionality and back end data stores before proceeding on to the details of the CIP middleware. Section 4 looks at the crosscutting functionality that was implemented across all of the CIP middleware services. Section 5 concludes by identifying some of the lessons learned on the project and noting what would be done differently if we were to do it all over again.

1.1 MER Operations

The MER operations (Ops) environment is stressful, to say the least. The turnaround time from downlink to uplink is approximately 14 hours, barely enough time for the teams of scientists and engineers to analyze the recent data, plan the next sols activities, and then generate and verify the command sequences before uplink radiation. To make matters worse, a sol is approximately forty minutes longer than a day is on Earth, so each day's activities, meetings, etc. occur a little later than they did the previous day. What this means, practically, is that the time of telemetry downlink, and all associated activities occurring afterwards, moved forward by approximately an eight-hour shift every other week, making it extremely difficult for mission personnel to maintain a sense of situational awareness about the mission. Operations proceeded in this fashion 24x7 for the duration of each rover’s life. The cumulative stress on Ops personnel makes it exceedingly difficult for them to remember when activities are to occur or where data products are located.

The total amount of data associated with the MER mission is quite large. The 1996 Mars Pathfinder mission, for reference, produced 2.3 gigabits of data in over 17,000 data products in the nineteen days the rover was active. With two rovers in operation, several GB of data was produced every day, including Experimental Data Records (EDRs) from the rovers themselves, Reduced Data Records (RDRs) as a result of EDR analysis, and the planning documents and command sequences produced within Ops. This volume of data not only needs to be archived, but also catalogued and tagged with metadata for easy indexing and retrieval. Furthermore, the interdependencies of the various teams make the challenges of cooperative and collaborative work more difficult than they would be otherwise. For example, if modifications are made to a given resource then others who depend on that information must be notified as quickly as possible. Subsequent modifications to data products ripple outwards through other resources and people as the awareness of the changes spreads.

1.2 General Requirements

As mentioned previously, CIP was intended to provide an enhanced sense of situational awareness to mission personnel. Of primary importance was to allow secure local and remote access to mission data products and schedules. The system was to be available 24x7, with better than a 99% availability over the course of the mission and handle a load of approximately 150 concurrent users. Furthermore, addressing the needs of a diverse user community including scientists, engineers, and managers posed a challenge in and of itself.

2 Development Approach

Here we provide a brief overview of our approach to developing the CIP middleware and mention the tools we used. The basic principles we followed were to (a) follow applicable industry standards, (b) utilize existing COTS software and (c) use available commercial development tools. Essentially we wanted to avoid reinventing any wheels, as we had some hard deadlines that could not be pushed back. The enterprise software standards we chose to employ were primarily J2EE and SOAP-based web services. Our application server was BEA's WebLogic 8.1. The development tools that we used included Borland's JBuilder Enterprise Edition version 9, the CVS source control system, and the ANT build utility.

CIP needed to be platform-independent, so we naturally chose to use Java throughout the system, with J2EE being used for the middleware. After evaluating several application servers, using criteria that included cost, performance, technical support, and industry ratings, we chose the WebLogic application server from BEA Systems, Inc.

Borland's JBuilder Enterprise Edition offered advanced editing and debugging facilities. It was well integrated with web services and with WebLogic. While building modules under JBuilder, it automatically invoked the appropriate WebLogic build utilities, such as the ones that generated the web services interfaces to the public EJB methods. As we describe later in this paper, we employed web services as the interface between the client applications and the middleware EJBs. The BEA utilities automatically generated the client stubs and the server code for the services. Outside of JBuilder, we used CVS for version control of our source files and ANT to do system builds.

3 CIP Architecture

In this section we provide an expose of the CIP architecture. The first two subsections outline the client application functionality and the back end data stores before presenting the CIP middleware and web services in detail. We conclude this section by examining how crosscutting functionality such as security and logging was implemented across all of the web services.

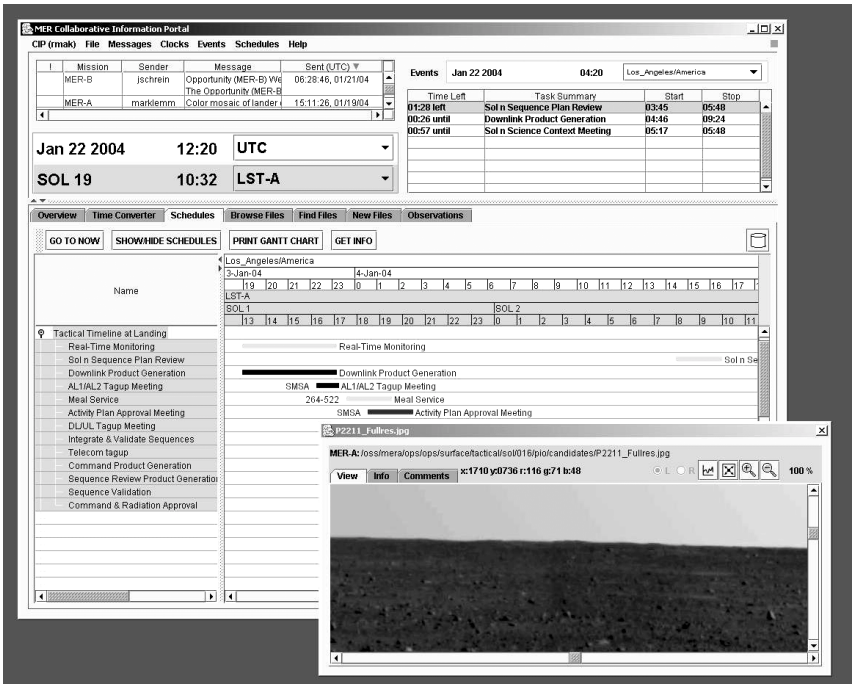

Fig. 1. The CIP client application, showing broadcast messages and clocks in the upper left, Event Horizon in the upper right, and tabbed pane for doing time conversions, displaying event and personnel schedules, browsing data files, searching for files, monitoring the arrival of new files, and displaying observations. In this screen shot, part of a tactical timeline schedule is visible with the time scale showing both Earth and Mars times. The auxiliary window is displaying an image selected by the data file browser

3.1 Client Application

CIP employs a three-tier client-server model in which the client is a desktop application and the server is a J2EE compliant middleware platform. The client application interface is divided into different information panels as shown in Figure 1. The server maintains user preferences and profile data in order to provide a consistent interface and behavior from one session to the next. The suite of tools within CIP facilitated increased communication, improved situational awareness and provided a greater exposure to mission information than would otherwise have been possible.

In essence, mission management, operations and science team members used CIP to retrieve the various data products, staffing and event schedules and maintain a sense of what was going on in the mission. In addition to integrating the various mission data repositories into a common interface, CIP provided custom tools for the following: Time keeping and conversion between Mars Local Solar Time for each rover (LST-A and LST-B) and time zones on Earth. Tracking of strategic and tactical events with an event horizon and schedule viewer that is linked to information about science teams and data products. Access to mission broadcast announcements and lastly, the ability to manage subscriptions to different types of notifications about new resources.

The CIP clocks maintain the correct time to within minute accuracy (due to network latency) and convert between different time zones on Earth as well as two Mars time zones LST-A and LST-B. An Event Horizon panel is also provided to display upcoming, ongoing and recent events, including communication windows, shifts and handovers, activity planning and approval meetings, communications windows, press conferences, etc.

The schedule viewer is an interactive tactical event timeline. The time zones supported by the scheduler are the same as those supported by the CIP Clocks. Tactical events are displayed in the schedule viewer, which can also display staffing information for mission personnel, press conferences, etc. The schedule is hyperlinked such that when a user clicks on an event, information about the associated teams, processes, and data products is displayed in a pop-up window.

The Browse Files and Find Files tabs provide tools for searching and browsing mission data products and resources by name, type, date, etc. In addition, the New Files tab allows users to subscribe to notification about active data products that they are interested in. The mission resources of interest include generated summary reports, rover health information, and strategic and activity plan summaries among other things. The data products reside on mission specific file systems, while annotations and other resource metadata are maintained in a metadata database. These repositories are described in more detail in the next subsection.

The broadcast announcements panel allows CIP users to communicate about mission announcements, news, events, and plans. Past messages are archived to a database and accessible through a message browser utility located in a pull down menu. Using the message browser, users can delete messages from their view or make important announcements 'sticky' so that they don't scroll off the screen as new messages appear.

3.2 Data Repositories

A critical aspect of the CIP application is being notified when new resources become available or when existing resources are modified. This allows the metadata cache to be kept up to date, in turn ensuring that mission personnel are using the most recent versions of resources to make important decisions. Maintaining a sufficient level of situational, group, and workspace awareness in a distributed system required a reliable event notification infrastructure.

For simplicity and responsiveness under extreme user loads, MER utilized several flat NFS file systems to store mission data products, hence the amount of control over the repository was limited when compared to other, more robust content management systems. To provide CIP with information about active resources, the Solaris NFS logging daemon, nfslogd, was used to trap file operations. Information is written to a log file that is monitored by a data acquisition daemon running on the CIP server.

When a given resource is active, a notification is sent to the CIP middleware in the form of a JMS message. The notifications contain enough information to update the CIP metadata repository appropriately. Subsequently, a notification was sent to any concerned clients about the update. The CIP metadata is maintained in an Oracle database on a dedicated server. This database also maintains schemas for scheduling information and archival storage of broadcast messages.

3.3 Middleware

CIP is a three-tier enterprise system based on the Java language. The client tier consisted of desktop applications developed with Swing components. The J2EE-conformant middleware tier consisted of Enterprise JavaBeans (EJB) and servlets running under the WebLogic application server. Web services were the interface between the client applications and the EJBs. The data repository tier included the mission file system, the Oracle databases, and data monitor and data loader utilities. The middleware EJBs used JDBC calls to access the databases. The Java Message Service (JMS) provided asynchronous messaging among the three tiers.

Figure 2 shows the CIP middleware architecture and its' relationship with the client application and back end data repositories. The middleware consisted of a number of independent services. Each service had a public interface implemented by a stateless session EJB that was the service provider. Each service provider had a SOAP processor to handle incoming web services requests and their responses. CIP used the HTTPS protocol to ensure secure communications between the client applications and the middleware.

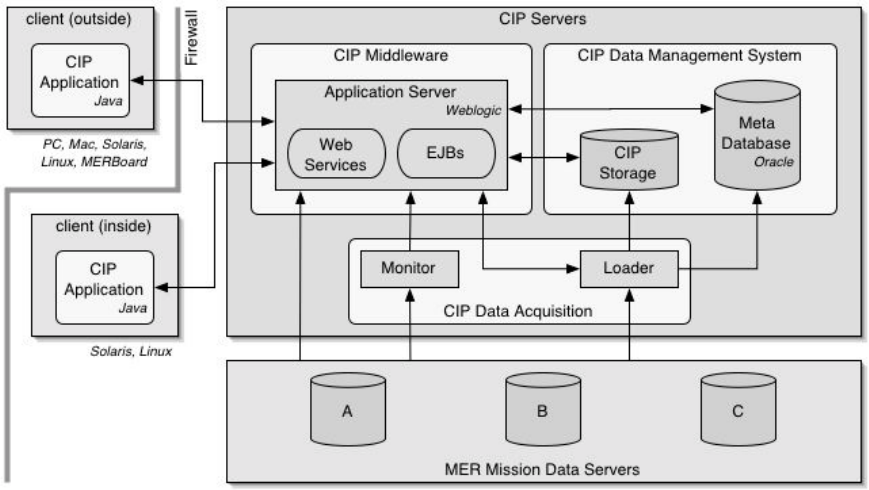

Fig. 2. The CIP middleware architecture and its' relationship to client applications (inside and outside the Ops firewall) and the back end data repositories. In the final deployed configuration, the CIP middleware, data acquisition and data management system ran on separate machines. The MER mission data servers were NFS mounted as shown

The following web services were exposed to the client application: User management, Time, Database query, File streaming, and JMS messaging. These services were chosen based on logical groupings of functionality present in the system requirements. The primary goals of the middleware included scalability, reliability, extensibility, and security. For the most part, these goals were fulfilled by our use of J2EE and web services over HTTPS. Our other goal for the middleware was that it remained invisible to the end client, giving each user the illusion of having direct and exclusive access to the data. It was evident that we had achieved this latter goal when users asked, "What server?", not realizing that they were connected to one in the first place.

User Management Service. The CIP User Management Service provides authentication and authorization services for CIP clients and is designed to be an independent vertical service that can be used for either purpose across serveral MER subsystems apart from CIP. The requirements for other MER subsystems to use the CIP User Management Service was inclusion of the user management service client stubs by way of a jar file. The clients also need to include a runtime client JAR file from Weblogic for using the WebLogic Server-provided implementation of the SSL client classes.

Time Service. It was important for everyone working on the MER mission to be able to answer the question "What time is it?". The mission ran on Mars time, and since a Sol is about 40 minutes different than an Earth day, regularly scheduled events drifted

later from day to day relative to Earth time. Moreover, two Martian time zones, one per rover were used extensively throughout the mission.

The CIP application displayed clocks that showed Mars and Earth times in various time zones chosen by the user. The CIP middleware supplied accurate times, which went over the Internet to the CIP applications. Due to network latencies, the times displayed by the CIP applications could be a few seconds off; we only guaranteed accuracy to the minute. The middleware obtained accurate Earth time via a Network Time Protocol server and computed the Mars times from the Earth time.

Query Service. The middleware query service accessed schedules and metadata stored in the Oracle databases of the data repository tier. CIP client applications displayed staff and event schedules, which they requested from the middleware. The applications also allowed users to browse data and images sent by the rovers. This information was categorized according to metadata fields, such as which rover, which sol, or which instrument. Users could also do searches based on metadata field values. The metadata was generated and loaded into the database by the data loader utility in the data repository tier.

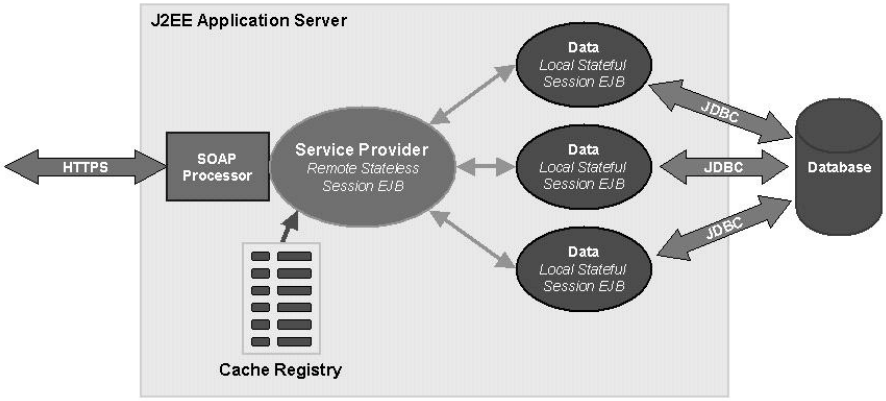

Fig. 3. The CIP Query Service

As shown in Figure 3, the query service used stateful session EJBs to perform the queries via JDBC calls. Each session EJB stored the query results. By keeping track of these stateful session EJBs in a registry – a hash table keyed by the query strings – the middleware was able to cache query results in memory. It was always much faster to respond to a data request from memory rather than by making an actual database query, especially since the database ran on a separate server.

Streamer Service A user of the CIP client application often requested downloads of data and images. The middleware responded to such a request by accessing the desired file from the mission file servers in the data repository tier.

The middleware transmitted each file in 64K blocks. The web services interface converted each block of binary data into ASCII – two characters per original byte – using base 64 notation. The block was then encrypted and sent over HTTPS to the requesting client application. The web services client stub decrypted the block and converted it back to binary. The application received all the blocks sequentially, and so it was simple for it to reassemble the data or image.

Despite all the data conversions and the encryption and decryption, files downloaded at the rate of around 100 MB per hour. Most of the data or image files were smaller than a few megabytes.

Message Service. The CIP Message Service provided JMS messaging capabilities between the CIP client and the middleware. The clearest example of this functionality is the broadcast announcements tool, which allowed managers to send messages out to the mission. Many of the CIP components, however, were able to take advantage of this framework so that the information they displayed was the freshest possible.

The Schedule Viewer subscribed to messages about modifications to the mission schedules. When a new schedule is uploaded to the server, or an existing schedule is modified, a message is published to 'schedules' topic. When the client receives the message, the display is updated to reflect that new scheduling information is available. Similarly, the New Files tab allows users to subscribe to active resources by their type and when they were last modified. When resources are modified or added to one of the mission repositories, an active resource notification message is published to a 'resources' topic and if a client has subscribed to notifications of that type, it is notified.

On the data acquisition side of things, the database loader subscribes to a 'monitor' topic, which carries messages about NFS operations on mission data products. This information is used to keep the metadata about the mission resources accurate and up to date. Thus, when the loader receives a notification that new data products are available it populates the metadata database with information about them.

The decoupling of event producer and event consumer components within the overall system is perhaps one of the greatest benefits of using messaging oriented architectures. This approach redeemed itself when it necessary to migrate the data acquisition components to another machine in order to lighten the load on the primary machine that clients connected to.

One of the difficult challenges in designing the CIP Message Service was in allowing multiple clients to connect using the same user ID. This situation would arise, for

example, whenever a user left an instance of the CIP client application running in their office or at home and then subsequently attempted to log in at a terminal in one of the science areas. This posed a problem because the application used durable subscriptions for several of the system services. As anyone familiar with the JMS specification will tell you, durable subscriptions require a unique client ID to be used consistently from one session to the next and only a single client can use this client ID at a given time. The solution was to use a client proxy on the server to connect to the JMS server in the J2EE container of the application server. In this way, when a user logged in more than once, we could detect that this was the case and manage the JMS subscriptions appropriately.

4 Crosscutting Functionality

In this section we take a look at the crosscutting functionality that was implemented across all of the middleware services.

4.1 Runtime Configuration

An important measurement of software reliability is how long it stays up and running. An application can unexpectedly crash, or system administrators can bring it down for maintenance. A common maintenance operation is to reconfigure an application to meet some change in operational usage or deployed configuration.

A key feature that allowed CIP to stay up and running for long periods (over 41 days at a time) was dynamic reconfiguration. CIP's middleware design allowed individual services to be *hot redeployable*. In other words, it was possible to reconfigure and restart a given service while the rest of the middleware web services (and CIP as a whole) continued to run without interruption.

To reconfigure a service, a system administrator first edited the service's configuration file and then redeployed the service. When the service restarts, it reads in the new configuration. Redeploying a service typically took only a few seconds, and often users did not notice any interruptions.

4.2 Security and Authorization

According to Sun Micro Systems J2EE specification there are two ways to control access to application resources using the J2EE architecture, declarative authorization and programmatic authorization. In the declarative authorization the application permission model is defined in a deployment descriptor. In the programmatic authorization the container makes access control decisions before dispatching method calls to a component. The CIP User Management Service uses programatic authorization. In this architecture, the J2EE container serves as an authorization boundary between the components it hosts and their callers

Authentication is the mechanism by which CIP clients prove that they are acting on behalf of specific users or systems. The authetication process for a CIP client is as follows. CIP client sends a login request by sending username and password on https (or http if the client is inside the Ops firewall). The User Management Service autheticates the credentials by interacting with the container (Weblogic server). If the authnetication is sucessful, the service generates an access token and retuns the token to the client. If the request fails the service returns a null token.

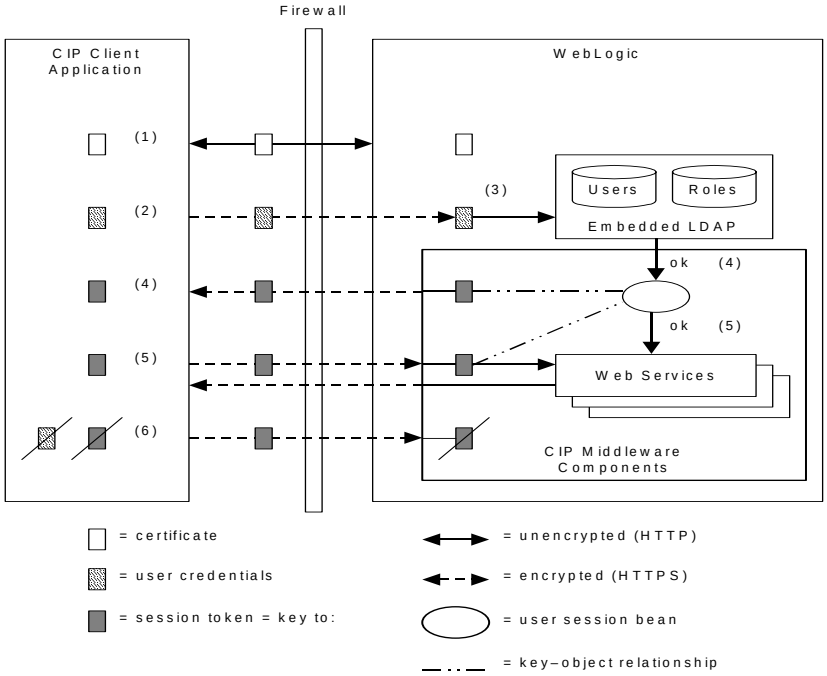

Fig. 4. CIP Authentication and authorization scheme in which (1) SSL certificate is exchanged, (2) user credentials are submitted over a secure channel, (3) user credentials are verified against the embedded LDAP store, (4) a user session EJB is created along with a corresponding session token, which is passed back to the client, (5) subsequent requests for middleware web services include the user session token, which is used by the middleware to look up the user session EJB and authorize the request against the associated roles and privileges, and (6) the expiration of the session token and removal of the user session EJB upon client logout. The actions in (6) may also occur if a CIP client is idle for too long and their session expires

Authorization is the process whereby the interactions between users and CIP services are controlled, based on user identity or other information. The user authorization for CIP is accomplised by the definition and enforcement of user roles. The authorization

process for a CIP client is as follows. When a CIP client requests a CIP middleware service, it supplies the login token obtained in a prior authetication request. The token is validated and then user is checked against the roles for granting permission to access the service requested. If the check is successful, the service request will be allowed to proceed, otherwise an authorization error will be returned to the client. The complete lifecycle of CIP authentication and authorization described above is depicted in Figure 4.

4.3 Logging and System Monitoring

Extensive run time logging and real time monitoring enhanced the middleware's reliability. The middleware web services logged every user request and it's subsequent processing within the system. For each request, the log entry contains a timestamp, the user's name, the named of the called method, details of the request, and key information about the results. By "mining" these logs, it was possible to deduce usage patterns and tune the system configuration accordingly. Figure 5 shows sample log entries.

```
2004-04-01 12:09:32,225 INFO : rmak: Metadata.query()
2004-04-01 12:09:32,230 DEBUG: SELECT file_view.* FROM MER_B.file_view
WHERE ((file_view.modified >= 1080806949117) AND (file_view.category =
'dataFile') AND (file_view.filename LIKE '/%/merb/ops/ops/surface%/%/rcam/%'
ESCAPE '\'))
2004-04-01 12:09:33,126 DEBUG: Records fetched: 0, skipped: 0
2004-04-01 13:50:06,816 INFO : rmak: Metadata.query()
2004-04-01 13:50:06,820 DEBUG: SELECT file_view.* FROM MER_B.file_view
WHERE ((file_view.seqnum = 66) AND (file_view.category = 'dataProduct') AND
(file_view.owner = 'opgs') AND (file_view.type LIKE '%/jpeg/MER-B' ESCAPE
'\'))
2004-04-01 13:50:10,073 DEBUG: Records fetched: 1, skipped: 0
2004-04-01 13:50:11,546 INFO : rmak: Metadata.getObjectsByParent()
2004-04-01 13:50:11,550 DEBUG: SELECT * FROM MER_B.file_view WHERE
(parent_pk = 16127) AND (category = 'dataFile')
2004-04-01 13:50:12,108 DEBUG: Records fetched: 5, skipped: 0
```

Fig. 5. Sample log entries for the CIP Metadata Query Service

A separate utility monitored the status of the middleware and graphically reported statistics such as memory usage and response times. Knowing the health of server at all times enabled us to correct any problems before they became serious. This monitoring was made possible by writing status information for each of the web services to a static data structure that was made available via inclusion on the application servers' main classpath. Figure 6 shows a screenshot of the graphical monitoring utility. In addition to the statistics tab shown, the Users tab displayed the users currently logged in, the file tab showed which files had recently been accessed

by users, and the cache tab showed the metadata that was currently cached from users interaction with the system.

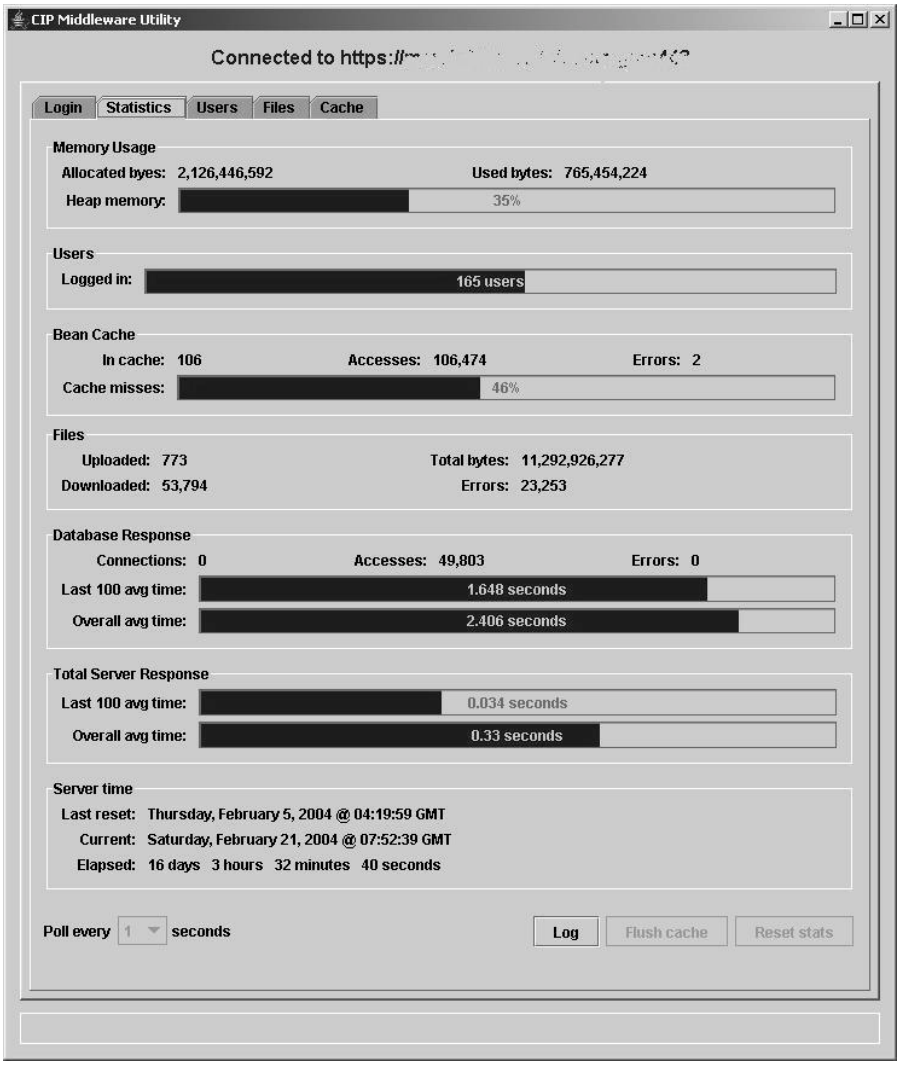

Fig. 6. The CIP Middleware monitoring utility, displaying the primary statistics tab. The other tabs displayed the currently logged in users, recently downloaded files and cached metadata

4. Conclusions

All things considered, CIP performed admirably throughout the MER mission and has received a significant amount of praise for the broad functionality it provided. The responsiveness of the system, its' robustness in the face of heavy user load and the overall availability of the system throughout the mission all met the original requirements and specifications. Further, the flexibility and ease of runtime configuration made it relatively straightforward to make last minute changes when necessary. Nevertheless, there are, of course, aspects of the design that we would do differently if we had to do it all over again.

When we first designed the middleware, a very complex data model was already in place that precluded the use of entity EJBs. Therefore, as described above, we used stateful session EJBs instead, and we implemented our own data caching algorithm. This became problematic, and we had to solve many thread synchronization problems. In hindsight, we should have simplified the data model and used entity EJBs. The lesson for future projects is to design the data model at the same time as the middleware to ensure that entity EJBs can be used.

After proving its worth during a series of Operational Readiness Tests at JPL, the mission managers deemed CIP to be mission critical. By then, it was too late for us to increase its reliability by clustering the middleware servers. While CIP turned out to be extremely reliable (> 99.7% uptime) despite its middleware running on a single server, we were obliged to monitor the system 24x7 over the course of the mission. The lesson for future projects is to always design the middleware to run on clustered servers. You can always turn off the clustering later if it isn't needed.

Despite the shortcomings recognized in hindsight, we feel that overall the CIP middleware architecture is an example of cutting edge technologies that ultimately overcame several difficult challenges. Primary among these is the way in which we were able to effectively unify multiple distinct web services into a single application platform.